Chemical Reaction Metaphor in Distributed Learning Environments

Lin, H.; Yang, Chunsheng

*Chemical Reaction Metaphor in Distributed Learning Environments **

Lin, H., and Yang, C.
May 2004

* published in the Proceedings of the International Conference on Industrial & Engineering. Applications of Artifical Intelligence & Expert Systems (IEA/AIE-2004). Ottawa, Ontario, Canada. May 2004. NRC 46551.

Chemical Reaction Metaphor in Distributed Learning Environments

Hong Lin[1] and Chunsheng Yang[2]

[1]Department of Computer & Mathematical Sciences, University of Houston-Downtown
1 Main Street, Houston, Texas 77002, USA
linh@uhd.edu
[2] Institute for Information Technology, National Research Council
1200 Montreal Rd, Ottawa, Ontario, Canada K1A 0R6
Chunsheng.Yang@nrc.gc.ca

Abstract. This paper presents an application of Chemical Reaction Metaphor (CRM) in agent-based distributed learning systems. The suitability of using CRM to model multi-agent systems is justified by CRM's capacity in catching dynamic features of multi-agent systems in an e-learning environment. A case study in course material updating demonstrates how the CRM based language, Gamma language, can be used to specify the architecture of the learning environment. Finally, a discussion on the implementation of Gamma language in a distributed system is given.

Keywords: Agent-oriented modeling, e-learning, program specification, very high-level languages

1. Introduction

Distributed learning is becoming a more and more prevailing method for conveying courses in recent years. The absence of the needs for classrooms and fixed time schedule adds to the flexibility in course delivery, which includes: virtual classroom, asynchronous mode teaching, and mobility.

To allow these advantages, however, software engineering is burdened with unprecedented challenges of implementing such a learning environment, which should be of the following main features: adaptive curriculum sequencing, problem solving support, adaptive presentation, student model matching. It is very difficult to develop a system that could meet all requirements for every level of educational hierarchy since no single designer of such a complex system can have full knowledge and control of the system. In addition, these systems have to be scaleable and provide adequate quality of service support [1]. This gives reason to finding a model that can catch the interactive and dynamic nature of e-learning systems. Such a model should be general enough to address common architectural issues and not be specific to design issues of a particular system. A direct benefit of such a model is expressiveness and extensibility --- changes in the domain knowledge would not require an intensive system-wide modification to alter the information and all objects that initiate actions based on that changing information.

2. Agent-Oriented Modeling and the Chemical Reaction Metaphor

The agent concept provides a focal point for accountability and responsibility for coping with the complexity of software systems both during design and execution [2]. It is deemed that software engineering challenges in developing large scale distributed learning environment can be overcome by an agent-based approach [3]. In this approach, a distributed learning system can be modeled as a set of autonomous, cooperating agents that communicate intelligently with one another. As an example, Collaborative Agent System Architecture (CASA) [4, 5] is an open, flexible model designed to meet the requirements from the resource-oriented nature of distributed learning systems. In CASA, agents are software entities that pursue their objectives while taking into account the resources and skills available to them.

We found that the dynamic nature of distributed agents in e-learning environments makes it an ideal object for modeling by Gamma languages [6-9]. The concurrency and automation of agents require that the modeling language does not have any sequential bias and global control structure. In addition, the dynamic nature and non-determinism of interaction between an agent and its environment are suited to a computation model with a loose mechanism for specifying the underlying data structure. Therefore, chemical reaction metaphor provides a framework for the specification of the behavior of an agent. For example, data, which move around the internet, can be well modeled by chemical resolution; and mobile agents, which are created dynamically and transferred from clients to servers, can be included in the environment variable of a higher-order Gamma configuration. This provides a mechanism for describing both inter-agent communications and agent migration.

3. Specifying Multi-Agent Systems in an E-Learning Environment

From the workflow model of the course development, we can build a collaborative system model that partitions the problem into one or more smaller tasks, which are tackled by corresponding agents. For example, let's examine the multi-agent system for course maintenance and recommendation that was designed in [10]. The online course materials are updated often in order to keep them as current as possible, esp. in some rapidly changing fields like 'computing and information systems'. Because of the complexity of the materials, and the short development cycles within which they are produced, the course instructor should make the necessary adjustments time by time for the benefit of the students. Whenever there is a significant change on the content of several designated web pages of online course materials, students who take the course should be notified by the course coordinator by e-mail. Figure 1 shows the conversation schemata for course maintenance.

The conversation model of the course material change notification consists of the following elements. For simplicity of illustration, we assume that a student who takes the course is in one of the 3 phases, numbered 1, 2, or 3. The interpretation of the phases is trivial and left undefined (For example, phase 1 might be the phase before the first exam, phase 2 the phase between the first exam and the second exam; and phase 3 the phase between the second exam and the final exam.) except that we assume only students who have passed the previous phase are allowed to enter the next phase. A course web page also bears a phase number, indicating to which phase its content is significant. Once a change is made to a web page, all students taking the course and whose phase number matches the phase number borne by the web page will be sent the link to that page.

- ***Notification Agent Control Client (NACC)***: The Notification Agent Control Client of an instructor or a student runs on his/her machine and allows him/her to control the behavior of the corresponding Notification Agent deployed in a distributed environment. In our system, NACC adds a student into the student database or removes him/her from the database, or changes the phase number the student is currently in.
- ***Notification Agent (NTFC)***: The basic function of the Notification Agent is to send e-mails to students taking the course according to the student profiles stored in a database when the course material has been significantly changed.
- ***Monitoring Agent (MNTR)***: The Web Change Monitoring Agent of a system administrator monitors a collection of course material URLs stored in a database. When the agent detects a significant change, it sends a message to the Notification Agent. Also, once a broken link is detected in the topic tree, it notifies the maintenance agent to either correct the link or delete the orphaned page.
- ***Student Information Agent (STIF)***: A Student Information Agent is designed for providing services about student information, such as providing an e-mail list for a course by automatically maintaining the email list of students taking a course; and maintaining the profile of each student.
- ***Maintenance Agent (MNTN)***: The maintenance agent provides proxy services to the instructor. It maintains the content of the topic tree.
- ***Topic Tree or Link Database (LINK)***: The course material is organized in the form of a topic tree. Each entry in the topic tree is a link to a web page.

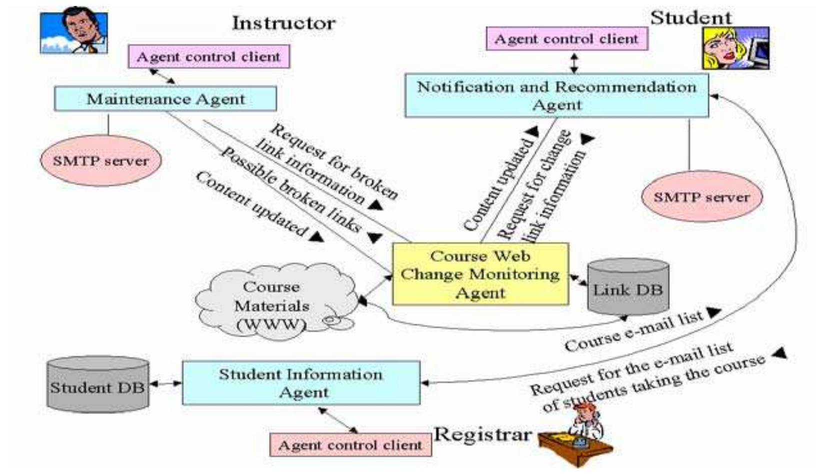

Figure 1: A conversation schemata for course maintenance

Let *INST* and *STUD* denote the multisets of instructors and students, respectively, and *I*, *S*, and *L* denote the instructor (We assume that there is only one instructor), the initial roll of the class, and the initial content of the course (in the form of the set of links), respectively, the following is the Gamma program that specifies the above system:

```
MAIN i S0 L0 = [P, NACC = [Q1, STUD = S0], NTFC = [Q2, STUD = S0, LINK = L0],
    MNTR = [Q3, LINK = L0],
    STIF = [Q4, STUD = S0], MNTN = [Q5, INST = {i}, LINK = L0]] where
    P = P1 + P2 + P3 + P4 + P5
        P1 = [Q1, STUD = S+{(s, 1, Ø)}]: NACC, [Q2, STUD = S', LINK = L]: NTFC
            → [Q1, STUD = S+{(s, 1, Ø)}]: NACC, [Q2, STUD = S'+{(s, 1, Ø)}, LINK
            = L]: NTFC ← (s, 1, Ø) ∉ S'
        P2 = [Q1, STUD = S+{(s, NULL, M)}]: NACC, [Q2, STUD = S', LINK = L]:
            NTFC → [Q1, STUD = S]: NACC, [Q2, STUD = S' - {(s, p, M)}, LINK =
            L]: NTFC
        P3 = [Q1, STUD = S+{(s, p, M)}]: NACC, [Q2, STUD = S'+{(s, p, M')}, LINK
            = L]: NTFC → [Q1, STUD = S+{(s, p, M')}]: NACC, [Q2, STUD = S'+{(s,
            p, M')}, LINK = L]: NTFC ← M ≠ M'
        P4 = [Q2, STUD = S, LINK = L+(l, p, normal)]: NTFC, [Q3, LINK = L'+{(l, p,
            changed)}]: MNTR → [Q2, STUD = S, LINK = L+(l, p, changed)]: NTFC,
            [Q3, LINK = L'+{(l, p, changed)}]: MNTR
        P5 = [Q1, STUD = S+{(s, 1, Ø)}]: NACC, [Q4, STUD = S']: STIF → [Q1,
            STUD = S+{(s, 1, Ø)}]: NACC, [Q4, STUD = S'+{(s, 1, Ø)}]: STIF ← (s,
            1, Ø) ∉ S'
        P6 = [Q1, STUD = S+{(s, NULL, M)}]: NACC, [Q4, STUD = S']: STIF → [Q1,
            STUD = S]: NACC, [Q4, STUD = S' - {(s, p, M)}]: STIF
        P7 = [Q1, STUD = S+{(s, p, M)}]: NACC, [Q4, STUD = S'+{(s, p+1, M)}]:
            STIF → [Q1, STUD = S+{(s, p+1, M)}]: NACC, [Q4, STUD = S'+{(s,
            p+1, M')}]: STIF
        P8 = [Q2, STUD = S+{(s, p, M')}, LINK = L]: NTFC, [Q4, STUD = S'+{(s, p,
            M)}]: STIF → [Q2, STUD = S+{(s, p, M')}, LINK = L]: NTFC, [Q4, STUD
            = S'+{(s, p, M')}]: STIF ← M ≠ M'
        P9 = [Q2, STUD = S+{(s, p, M')}, LINK = L]: NTFC, [Q4, STUD = S'+{(s,
            p+1, M)}]: STIF → [Q2, STUD = S+{(s, p+1, M')}, LINK = L]: NTFC,
            [Q4, STUD = S'+{(s, p+1, M)}]: STIF
        P10 = [Q2, STUD = S, LINK = L]: NTFC, [Q5, INST = I, LINK = L']: MNTN →
            [Q2, STUD = S, LINK = L']: NTFC, [Q5, INST = I, LINK = L']: MNTN ←
            L ≠ L'
        P11 = [Q3, LINK = L]: MNTR,, [Q5, INST = I, LINK = L']: MNTN → [Q3, LINK
            = L']: MNTR, [Q5, INST = I, LINK = L']: MNTN ← L ≠ L'
    Q1 = Enrl + Drop
        Enrl =          (s, 1, Ø): STUD ← Enroll(s)
        Drop = (s, p, M): STUD → (s, NULL, M) ← Drop(s)
    Q2 = Updt ∘ Emal
        Emal = (l, p, changed): LINK, (s, p. M): STUD → (l, p, changed): LINK, (s, p.
            M+{l}): STUD ← l ∉ M
        Updt = (l, p, changed): LINK → (l, p, normal): LINK
```

Q3 = (*l, p*, **normal**): *LINK* → (*l, p*, **changed**): *LINK* ← *Modified*(*I*)
Q4 = (*s, p, M*): *STUD* → (*s, p+1, M*): *STUD* ← *Pass*(*s, p*)
Q5 = *AddInst* + *AddLink* + *Chng* + *Updt*
AddInst = *i: INST* ← *AddInst*(*i*)
AddLink = *i: INST* → (*l, p*, **normal**): *LINK, i: INST* ← (*l, p*) = *AddLink*(*l, i*)
Chng = (*l, p*, **normal**): *LINK, i: INST* → (*l', p*, **changed**): *LINK, i: INST* ← *l'*= *Change*(*l, i*)
Updt = (*l, p*, **broken**): *LINK, i: INST* → (*l', p*, **normal**): *LINK, i: INST* ← *l'*= *Update*(*I, i*)

In this program, constants are written in boldface words. Each student record is a tuple (*student, phase, mailbox*) where *student* is the name of the student, *phase* the phase number where the student is in, and *mailbox* the mailbox of the student, which is a multiset of email messages. Each entry of the link database is also a tuple (*link, phase, status*) where *link* is the link to the web page in the topic tree, *phase* the phase number this page is designed for, and *status* the status of the page, which can be either **normal, changed, or broken**. Boolean functions *Enroll*(*s*) and *Drop*(*s*) return whether student *s* is enrolled in the class or wants to drop. *Modified*(*l*) function returns whether a particular web page pointed to by link *l* has been modified or not. *Pass*(*s, p*) function finds out whether student *s* has passed phase *p* or not. *Add*(*l*, *i*) function indicates whether instructor *i* wants to add page pointed to by link *l* into the link database or not. *Change*(*l, i*) function returns the link to the changed page whose original is pointed to by *l. Update*(*l, i*) function updates the broken link *l* and returns the corrected link.

The program consists of configurations in two levels: the *MAIN* configuration in the higher level and all other configurations in the lower level. Program *P* in *MAIN* configuration exchanges elements of the multisets in the environments of the lower-level configurations.

This example shows how Gamma language expresses the architecture of a multi-agent system succinctly. With the underlying computing model, we do not need to consider the specifications of nonessential features of the system, e.g., the number of program units, connection links for communications, and organizations of data, and therefore can focus on the specification of the overall architecture. It catches the way program units interact with one another and local computations, such as the implementations of those local functions, are left to the subsequent design phase.

The specification of the overall system benefits the subsequent design phases because details of the system can be added into the system in an accumulative fashion. The following section describes the specification of individual program units.

4. From Architecture to Building Blocks

A systematic design strategy was proposed in [11], in which Gamma specification of an agent system can be implemented in a hierarchical running environment composed of nodes in different levels of a tree. Interactions among agents can be implemented in a unified mechanism for synchronization. In this scheme, each configuration in the Gamma specification is implemented as a node. The overall architecture of the system is a tree structure, which expands and shrinks dynamically. A node only communicates with another node in the immediate upper or lower level. Interfaces between nodes specify the local conditions that may cause an action in the upper level. The actions in the upper level (in which nodes are called controlling nodes) can be creating/deleting nodes in the lower level or transforming the states of nodes in the lower level by data transfer.

The specification of the type of a node is composed of the module name, declarations of environment variables, imported variables, exported variables, and a body block consisting of sequentially executed statements.

```
process name(parameter-list)
    environment     Local environment variables
    import          Imported variables
    export          Exported variables
    begin
                    Statements
    End
```

Variables represent data sets. We leave the data structure for variables unspecified to maintain high-level abstraction. Imported variables and exported variables are written in the form of Module.Variable. If Module is omitted, the variable is identical to the local variable. Imported variables store values received from the nodes in the immediate lower or upper level while exported variables stores the values that are sent to the node in immediate lower or upper level. Both imported variables and exported variables can be interpreted as set of channels through which data are exchanged between nodes in adjacent levels. There is a separated channel established for each node. To maintain a high level of abstraction, we do not distinguish channels for different nodes. Instead, we use operator X.node to find out the sending node through variable X. Channels are automatic objects, which means imported channels receive messages whenever a send action is initiated by another node and exported channels send messages whenever data are available. To send a message to another node, we only need to use add action to add data items into the exported variables. Although we may use the same variable in both the import and export section, incoming data and outgoing data are distinguished by default. That means that outgoing data are never used in local computation.

Parameter list is used to pass initial values to the process when the process of the particular module is created. There are four actions that can be performed by a process:

- Add(variable, data): add data into variable
- Delete(variable, data): delete data from variable
- Select(variable): select an element of the data set represented by variable
- element.#n: projection operation --- extract the nth value of the tuple denoted by element

Statements in the body block of a module can be an add/delete action, a branching statement, or a looping statement. We omit the description of branching statements because they are not used in this program. The looping structure has the following syntax:

```
do   cond1 -> statement1;
     cond2 -> statement2;
     …
     condn: -> statementn;
od
```

The semantics of the looping statement is: cond1, …, condn are tested and one of the statements whose corresponding condition is tested to true is executed non-deterministically. Conditions are tested repeatedly until none of the conditions evaluates to true and the control is then transferred to the statement that follows the do statement.

The modules designed for the course maintenance program in the previous section is described in the following:

```
process NACC(STUD firstRoll)
environment
          STUD roll = firstRoll;
import
          STUD roll;
export
          STUD roll;
begin
          do   Enroll(s)       Add(roll, (s, 1, Ø));
               s = Select(roll), Drop(s)          Delete(roll, s), Add(roll, (s, NULL, s.#3));
          od
end

process NTFC(STUD firstRoll, LINK origLink)
   environment
          STUD roll = firstRoll; LINK link = origLink;
   import
          LINK link;
   export
          STUD roll; LINK link;
   begin
          do    l = Select(link), l.#3 = “changed” →
                    do s = Select(roll), l ∉ s.#3 → Add(s.#3, l);od
          do    l = Select(link), l.#3 = “changed” → Delete(link, l), Add(link, (l.#1, l.#2,
                    “normal”));
          od
   end

process MNTR(LINK origLink)
   environment
          LINK link = origLink;
   import
          LINK link;
   export
```

```
            LINK link;
    begin
            do   l = Select(link), Modified(l) → Delete(link, l), Add(link, (l.#1, l.#2, "changed"));
            od
    end

process STIF(STUD firstRoll)
    environment
            STUD roll = firstRoll;
    import
            STUD roll;
    export
            STUD roll;
    begin
            do   s = Select(roll), Pass(s.#1, s.#2) → Delete(roll, s), Add(link, (s.#1, s.#2 + 1,
                        s.#3));
            od
    end

process MNTN(INST initInst, LINK origLink)
    environment
            INST inst = initInst;
            LINK link = origLink;
    import
            LINK link;
    export
            LINK link;
    begin
            do   AddInst(i) → Add(inst, I);
                 i = Select(inst), l = AddLink(l, i) → Add(link, (l.#1, l.#2, "normal");
                 i = Select(inst), l = Select(link), l.#3 = "normal", l' = Change(l, i) →
                       Delete(link, l), Add(link, (l', l.#2, "changed"));
                 i = Select(inst), l = Select(link), l.#3 = "broken", l' = Update(l, i) →
                       Delete(link, l), Add(link, (l', l.#2, "normal"));
            od
    end

process MAIN(INST initInst, STUD firstRoll, LINK origLink)
    environment
            NACC nacc; NTFC ntfc; MNTR mntr; STIF stif; MNTN mntn;
    import
            STUD nacc.rollNacc, ntfc.rollNtfc, stif.rollStif;
            LINK ntfc.linkNtfc, mntr.linkMntr, mntn.linkMntn;
    export
            STUD nacc.rollNacc, ntfc.rollNtfc, stif.rollStif;
            LINK ntfc.linkNtfc, mntr.linkMntr, mntn.linkMntn;
    begin
            Add(nacc, NACC(firstRoll));
            Add(ntfc, NTFC(firstRoll, origLink));
            Add(mntr, MNTR(origLink));
            Add(stif, STIF(firstRoll));
            Add(mntn, MNTN(initInst, origLink));
```

```
        do    s = Select(nacc.rollNacc), s.#2 ≠ NULL, s ∉ ntfc.rollNtfc → Add(ntfc.rollNtfc,
                  s);
              s = Select(nacc.rollNacc), s.#2 = NULL, s' = Select(ntfc.rollNtfc), s.#1 = s'.#1
                  → Delete(nacc.rollNacc, s), Delete(ntfc.rollNtfc, s');
              s = Select(nacc.rollNacc), s' = Select(ntfc.rollNtfc), s.#1 = s'.#1, s.#2 = s'.#2,
                  s.#3 ≠ s'.#3  → Delete(nacc.rollNacc, s), Add(nacc.rollNacc, s');
              l = Select(ntfc.linkNtfc), l' = Select(mntr.linkMntr), l.#1 = l'.#1, l.#2 = l'.#2,
                  s.#3 ≠ s'.#3 → Delete(ntfc.linkNtfc, l), Add(ntfc.linkNtfc, l');
              s = Select(nacc.rollNacc), s.#2 ≠ NULL, s ∉ stif.rollStif → Add(stif.rollStif, s);
              s = Select(nacc.rollNacc), s.#2 = NULL, s' = Select(stif.rollStif), s.#1 = s'.#1 →
                  Delete(nacc.rollNacc, s), Delete(stif.rollStif, s');
              s = Select(nacc.rollNacc), s' = Select(stif.rollStif), s.#1 = s'.#1, s.#2 + 1 = s'.#2,
                  s.#3 = s'.#3 → Delete(nacc.rollNacc, s), Add(nacc.rollNacc, s');
              s = Select(ntfc.rollNtfc), s' = Select(stif.rollStif), s.#1 = s'.#1, s.#2 = s'.#2, s.#3
                  ≠ s'.#3 → Delete(stif.rollStif, s'), Add(stif.rollStif, s);
              s = Select(ntfc.rollNtfc), s' = Select(stif.rollStif), s.#1 = s'.#1, s.#2 + 1 = s'.#2,
                  → Delete(ntfc.rollNtfc, s), Add(ntfc.rollNtfc, (s.#1, s'.#2, s.#3));
              s = Select(ntfc.rollNtfc), s' = Select(stif.rollStif), s.#1 = s'.#1, s.#2 + 1 = s'.#2,
                  → Delete(ntfc.rollNtfc, s), Add(ntfc.rollNtfc, (s.#1, s'.#2, s.#3));
              l = Select(ntfc.linkNtfc), l ∉ mntn.linkMntn → Delete(ntfc.linkNtfc, l);
              l = Select(mntn.linkMntn), l ∉ ntfc.linkNtfc → Add(ntfc.linkNtfc, l);
              l = Select(mntr.linkMntr), l ∉ mntn.linkMntn → Delete(mntr.linkMntr, l);
              l = Select(mntn.linkMntn), l ∉ mntr.linkMntr → Add(mntr.linkMntr, l)
        od
    end
```

Note that higher-order operations remain in the module level. This makes the specification of the system closer to actual program. Also note that the transformation from Gamma specification to module specification can well be automated. Further transformation from module specification to programs in concrete language can be facilitated. The specification in the module level still focuses on generic process behavior. Data structures are left unspecified. Further refinement of the specification should include the use of data structures to organize the data sets. Therefore the Select operation can be implemented by an algorithm designed in accordance with the data structure. Another refinement would be the implementation of the data exchange channels.

5. Conclusions and Future Work

We propose a method for specifying a multi-agent system by using Gamma language. We find that in chemical reaction metaphor, architectural properties of a multi-agent system can be expressed succinctly and precisely. Through the case study, we demonstrate the usefulness of this method in the design of a multi-agent e-learning environment.We present a method for transforming the Gamma specification of the agent system into the specification in a module language, in which higher-order multiset operations are removed. This paves the way for implementing the specified system by using a sequence of program transformation. In the future, we will be working on the automation of the program transformation process and the refinement of module specifications by introducing data structures into the program.

References

1. Vouk, Mladen A., Donald L. Bitzer and Richard L. Klevans, Workflow and End-User Quality of Service Issues in Web-Based Education, IEEE Trans. on Knowledge and Data Engineering, 11, (4); July/August 1999, pp. 673-687
2. Yu, Eric, Agent-Oriented Modelling: Software Versus the World, Agent-Oriented Software Engineering AOSE-2001 Workshop Proceedings. LNCS 2222. Springer Verlag. 206-225.
3. Vassileva J., Deters R., Greer J., MaCalla G., Kumar V., Mudgal C., (1998) A Multi-Agent Architecture for Peer-Help in a University Course, Proceedings of the Workshop on Pedagogical Agents at ITS'98, San Antonio, Texas, 64-68
4. Flores, R.A., Kremer, R.C., & Norrie, D.H. An Architecture for Modeling Internet-based Collaborative Agent Systems, in T. Wagner & O.F. Rana (Eds.), Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, LNCS1887, Springer-Verlag, 2001, 56-63.
5. Lin F. O., Norrie D. H., Flores, R.A., & Kremer R.C. Incorporating Conversation Managers into Multi-agent Systems, in M. Greaves, F. Dignum, J. Bradshaw & B. Chaib-draa (Eds.), Proc. of the Workshop on Agent Communication and Languages, 4th Inter. Conf. on Autonomous Agents (Agents 2000), Barcelona, Spain, June, 3-7, 2000, pp. 1-9.
6. Banatre, J.-P., & Le Metayer, D. (1990). The Gamma model and its discipline of programming. Science of Computer Programming, 15, 55-77.
7. Banatre, J.-P., & Le Metayer, D. (1993). Programming by multiset transformation, CACM, 36(1), 98-111.
8. Banatre, J.-P., & Le Metayer, D. (1996). Gamma and the chemical reaction model: ten years after. in: Andresli, J.M., & Hankin, C. (eds.), Coordination Programming: Mechanisms, Models and Semantics, Imperial College Press.
9. Le Metayer, D. (1994). Higher-order multiset processing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 18, 179-200.
10. Lin, F. O., Lin, H., & Holt, P., A Method for Implementing Distributed Learning Environments, Proc. 2003 Information Resources Management Association International Conference, May 18-21, 2003, Philadelphia, Pennsylvania, USA, 484-487.
11. Lin, H., A Language for Specifying Agent Systems in E-Learning Environments, in Fuhua Oscar Lin (eds.)*: Designing Distributed Learning Environments With Intelligent Software Agents*, Idea Group Inc., to appear.